REMARKS

Claims 1-27 pending in the application. Claims 1-27 stand rejected.

Claim 1 has been amended in this Response. No claims have been added. No new matter has been added. Applicant respectfully requests reconsideration of the pending claims in light of the following remarks.

A. Rejections of Claims 1,10, 16, 20, 22-24 and 26 Under 35 U.S.C. § 103

In the Office Action dated January 14, 2005, the Examiner rejected Claims 1, 10, 16, 20, 22-24 and 26 under 35 U.S.C. § 103(a) as being unpatentable over Huang, et al. (U.S. Patent No 6,151,582) (**HU**), in view of Rumbaugh, et al. (Object oriented modeling and Design, 1991) (**RU**). Applicant respectfully submits this rejection is traversed.

The Examiner states that **HU** teaches "a simulation system used by an operator and including a source of input data, a display, and a simulator adapted to be executed by a processor and generating a set of simulation results during the execution in response to the input data ..., an organizing and managing system..., one or more of the sets and supersets of test data..., editing means..., and the simulator generating a set of simulation results...." The Examiner further states that **HU** "teaches a case manager adapted for storing a plurality of sets and supersets of test data files, the sets and supersets of test data files being stored in the case manager in the form of a tree like structure." The Examiner acknowledges, however, that "**HU** does not expressly teach the sets and supersets of test data files being stored in the case manager in the form of a hierarchical non-conventional tree like structure having a root and one or more leaves, the tree like structure being non-conventional in that one or more of the supersets underlie corresponding ones of the sets

in the tree-like structure, such that one or more of the sets is situated between the root and the corresponding superset."

The Examiner asserts that "**RU** teaches the sets and supersets of test data files being stored in the case manager in the form of a hierarchical non-conventional tree like structure, having a root and one or more leaves, the tree like structure being non-conventional in that one or more of the supersets underlie corresponding ones of the sets in the tree-like structure, such that one or more of the sets is situated between the root and the corresponding superset...." (Office Action, January 14, 2005, lines 8-13, p. 4) Applicant agrees that **RU** teaches the sets and supersets of test data files being stored in the form of a hierarchical tree like structure having a root and one or more leaves, but respectfully disagrees with Examiner's assertion that the tree like structure of **RU** is non-conventional and respectfully disagrees with Examiner's assertion that **RU** teaches that one or more of the supersets underlie corresponding ones of the sets in the tree-like structure, such that one or more of the sets is situated between the root and the corresponding superset...."

1. **Both the present invention and Reference RU Use Data Hierarchy, Not Linked Lists.**

In responding to Applicant's previous amendments and arguments, the Examiner in the instant Office Action states:

> "Database organization is conceptual, all done by linked lists starting from one element and proceeding to other elements using one or more pointers associated with each element. The links indicate that the data in the linked sets are related. While each record can have one preceding record and one succeeding record, there is no constraint on the number of elements at each level of the

structure. There is no constraint in the database to limit the amount of data in various sets connected by the links and the type of data in the sets. Therefore the argument that superset is above the set or superset is below the set is a design choice for a particular application; however, in the database organization any amount of data or any type of data may be associated at any level of the data organization. Similarly, the data hierarchy is also conceptual. The database does not use data hierarchy since linked lists are used." (Office Action, January 14, 2005, paragraph 7.1, pp. 32-33, emphasis added.)

Applicant believes that the above, underlined statement is directly contradicted by the Examiner elsewhere in the Office Action, where Examiner asserts that **RU** does teach use of a "hierarchical non-conventional tree like structure." (Office Action, January 14, 2005, line 10, p. 4) Both **RU** and the instant invention use hierarchical, tree like structures. If despite Examiner's apparent acknowledgment of this, Examiner is still asserting that the present invention uses linked lists and is not hierarchal, Applicant respectfully points out that Examiner's assertion is totally unsupported by the specification and the claims of the instant application. By contrast, Applicant's assertion that the database does use data hierarchy finds ample support in the specification and is recited in the claims, as previously amended.

Webster's New World Dictionary of Computer Terms, compiled by Donald Spencer, pub. by MacMillan, USA, 1994 ("Webster's") (at p. 274) defines a "hierarchical model" as a "database model in which each object is of a particular hierarchy in a tree structure." (Applicant has attached the potions of Webster's referred to herein for the Examiner's convenience.)

Claim 1 was amended in this response to correct typographical errors inadvertently introduced in a previous amendment. No new matter has been added. Claim 1 was previously amended to clarify that the non-conventional tree like structure of the present invention is hierarchical in nature and to clarify the hierarchical nature of such tree structures:

> 1. In a simulation system used by an operator and including a source of input data, a display, and a simulator adapted to be executed by a processor and generating a set of simulation results during the execution in response to said input data, an organizing and managing system operatively interconnected between the source of the input data and said simulator and said display, comprising:
>
> a case manager adapted for storing a plurality of sets and supersets of test data files, said sets and supersets of test data files being stored in said case manager in the form of a hierarchical, non-conventional tree like structure, having a root and one or more leaves, the tree like structure being non-conventional in that one or more of said supersets underlie corresponding ones of said sets in said tree like structure, such that one or more of said sets is situated between the root and the corresponding superset, and further comprising one or more of said sets and said supersets of said test data files adapted to be selected by said operator; and
>
> editing means responsive to said one or more of said sets and said supersets of said test data files selected by said operator via said case manager and responsive to said input data for editing said test data files and said input data in response to editing actions taken by said operator and generating a set of edited test data files, said simulator generating said set of simulation results during the execution of said simulator in response to said set of edited test data files.

This previously-submitted amendment is fully supported by the application as filed. See, for instance, Specification as filed, Figure 21 and page 48, paragraph 00167: "The Case/Project Manager 46c1 layout (or 'window screen display') consists of a Menu Bar, Icons, and the current Project displayed as a hierarchical 'tree like structure.'" Emphasis added. Other explicit uses of the word "hierarchy" or "hierarchical" in the

specification occur at page 39 in paragraph 116, on page 51, paragraph 180 and on page 57, paragraph 225.

By contrast, there is no use of the words "linked list" in the application. Note also that paragraph [0168] of the substituted Specification at page 48 recites:

> "[0168] Nomenclature
> 1. Base - first simulator run of the current project
> 2. Case - any subsequent simulator run where the grid geometry has been changed from its parent
> 3. Scenario - any subsequent simulator run where the grid geometry remains the same as its parent"

(Emphasis added.) Likewise, paragraph [0172] of the Substituted Specification at page 49 recites:

> "[0172] Case
> 1. About - panel to show/enter case details
> 2. View - views input and output files associated with selected case/scenario
> 3. Load - loads selected case/scenario
> 4. Load As - loads selected case/scenario as a new case/scenario
> 5. Create - creates a new case/scenario from an existing simulator run
> 6. Delete - removes selected case/scenario and all children from project"

(Emphasis added.) These references to parent-child relationships further evidence the hierarchical nature of the present invention. Linked lists do not have parent-child relationships. At page 85, Webster's defines "children" as "In hierarchical databases, those data elements that are linked in a subordinate fashion to a parent element." (Emphasis added.) The database of the present invention is hierarchical.

Referring to Fig. 3.23 of **RU**, the figure is entitled "A multi-level inheritance hierarchy with instances." (Emphasis added.) Thus the database of Fig. 3.23 is also hierarchical.

Both the reference and the present invention are hierarchical, and Applicant respectfully asserts that neither uses linked lists.

2. **The Present Invention as Recited by Claim 1 is Patentably Distinct from the Cited References.**

The Examiner asserts "**RU** depicts a superclass below its corresponding class, the superset containing information from outside the set of higher entry as shown in Fig. 3.23." (Office Action at p. 32). The Examiner is not specific as to what exactly he is referring to within Fig. 3.23. Accordingly, Applicant is unsure as to whether the Examiner is contending that a superclass is shown below a subclass because the page can be turned upside down or whether Examiner is defining a superclass as "containing information outside the set of higher entry." Both possibilities are addressed and respectfully refuted below.

a. **A subclass may have information not specified in its superclass – but it is still a subclass with respect to that superclass.**

Taking the last possibility first, as **RU** states at page 39 near the end of the second full paragraph, "[e]ach subclass not only inherits all the features of its ancestors, but adds its own specific attributes and operations a well. For example, *Pump* adds attribute flow rate, which is not shared by other kinds of equipment." (Emphasis added.) That does not make *Pump* a superclass below *Equipment* in Fig. 3.23. *Equipment* is still the superclass and *Pump* its subclass, even though Pump arguably contains "information outside the set of higher entry." The key, as the authors of **RU** points out, is the relationship between class and subclass: "The class being refined is called the *superclass* and each refined version is called a *subclass*." **RU** at 39, in first full paragraph. (This corresponds, respectively, to the "supersets" and "sets" of the instant application.)

Accordingly, Applicant respectfully submits that if this is what Examiner was referring to as citing that "**RU** depicts a superclass below its corresponding class, the superset containing information from outside the set of higher entry...," Examiner's assertion has been refuted. Fig. 3.23 of **RU** depicts a conventional heirarchial tree like structure having a root and one or more leaves, but does not depict "the tree like structure being non-conventional in that one or more of said supersets underlie corresponding ones of said sets in said tree like structure, such that one or more of said sets is situated between the root and the corresponding superset...."

In **RU**, the "superclass is connected by a line to the apex of the triangle. The subclasses are connected by lines to a horizontal bar attached to the base of the triangle." (**RU** at p. 39 in third full paragraph, emphasis added.) Note that in Figure 3.23 of **RU**, the superclasses are connected in each instance by a bar to the apex (or top) of a triangle and its subclasses are connected by a bar to the base of the triangle. "For convenience, the triangle can be inverted and the subclasses can be connected to both the top and bottom of the bar, but if possible, the superclass should be drawn on top and the subclasses on the bottom." (**RU** at p. 39, emphasis added.) Using whether a bar points to a base of a triangle or to its apex, one can identify all subrclasses (connected to the base of the triangles) and their respective superclasses (connected to the apex) in Figure 3.23.

Hierarchical tree like structures, such as those used by both Fig. 3.23 of **RU** and the instant invention, are understood by those of ordinary skill in the art to have a "root" commonly shown at the top of the page and "leaf nodes" or "leaves" commonly shown at the bottom of the page. Webster's at p. 502 defines "root" as "The top element or node in a tree diagram, from which branches extend eventually to leaf nodes." "Leaf" is defined

at p. 327 as "The terminal node of a tree diagram." In other words, the convention, which **RU** follows, is to show the highest superclass or the "root" at the top of the page. Each subclasses is depicted "below" its respective superclass, that is a subclass is depicted between its superclass and the leaf nodes of the subclass. A subclass is thus on the leaf node side of its superclass. (Of course, the subclass may itself be a terminal leaf node.)

Fig. 3.23 follows out this convention with the highest order of superclass being "Equipment" at the top of the page and the leaf nodes being towards the bottom of the page. The copy of Figure 3.23 submitted herewith has been marked to indicate the root ("Equipment") and the leaf nodes, being the different types of pumps and tanks. Specifically, the leaves or leaf nodes of Fig. 3.23 include centrifugal pump, diaphragm pump and plunger pump (for the pump branch) and spherical tank, pressurized tank, floating roof tank (for the tank branch), with heat exchanger having no subclasses, so that it is itself a leaf node. *Centrifugal pump* is a subclass (and terminal leaf node) of its superclass *Pump*, which is itself a subclass of its superclass (and root) *Equipment*. **RU** does not depict *Centrifugal pump* on the root side of *pump*; if it did, *Centrifugal pump* would be situated between *Pump* and *Equipment.* It is not. *Equipment* is not shown situated between any of its subclasses (*Pump*, *Heat exchanger* and *Tank*) and corresponding leaf node(s) of the subclass. Unlike the present invention, Figure 3.23 does not depict any superclass being "below" (meaning "on the leaf node side of") a subclass or a subclass "above" (meaning "on the root side of") its superclass.

b. **Applicant amended Claim 1 to eliminate confusion as to what "below" means.**

The "above" and "below" language can be confusing and as the Examiner has noted, if a superclass is depicted below a subclass, the paper could be turned upside down, which might confuse one not familiar with the drawing conventions. Claim 1 was previously amended to eliminate the point of confusion about what "above" or "below" means in this context by using the root and leaves terminology instead.

If the diagram with a tree like structure conventionally depicted (root at the top, leaf nodes at the bottom) such a Fig. 3.23 is turned upside down, there will still be the same root (in the case of Fig. 3.23, *Equipment*, now at the bottom of the page) and one or more terminating leaf nodes. For the *Pump* branch of Fig. 3.23, *Centrifugal pump*, *Diaphragm pump* and *Plunger pump* would now be towards the top of the page, but would still be leaf nodes. The relative positions of the root, leaves, sets and supersets will be unchanged, no matter which direction the page is turned.

If, in Fig. 3.23, *Centrifugal pump* had an additional line leading to the base of an inverted triangle, with the apex of the triangle connected to a new class called, say, *Electric-powered equipment*, then *Centrifugal pump* would be a subclass to both *Pump* and *Electric-powered equipment* as it refines both classes. In that case, subclass C*entrifugal pump* would be situated between a superclass *Electric-powered equipment* and its root, *Equipment*. Fig. 3.23 shows no such configuration. **RU** teaches away from the present invention.

By contrast, the instant invention allows for one or more sets to be situated between their corresponding supersets and the root node. In other words, one or more supersets are situated between their corresponding sets and a leaf node, or a superset

could be a leaf node for a set. This, Applicant respectfully submits, is novel over the cited references. Accordingly, it is respectfully submitted that claim 1 is patentable over the combination of **HU** and **RU**.

3. **Like Claim 1, Claims 10, 16, 20, 22, 23, 24, and 26 Are Patentably Distinct from the Cited References.**

Like claim 1, claims 10, 16, 20, 22, 24 and 26 have been previously amended to recite the hierarchical, non-conventional tree like structure, having a root and one or more leaves, the tree like structure being non-conventional in that said supersets underlying corresponding ones of said sets in said tree like structure, such that one or more of said sets is situated between the root and the corresponding superset...." Accordingly, claims 10, 16, 20, 22, 24, and 26 are thus felt to be likewise patentably distinct over the combination of **HU** and **RU**.

Claim 23 depends from claim 22 and contains all of its limitations as amended. Accordingly, Applicant respectfully submits that this rejection has also been traversed with respect to claim 23.

Accordingly, Applicant respectfully submits that this rejection has been traversed and requests reconsideration and allowance of claims 1, 10, 16, 20, 22, 23, 24 and 26.

B. Rejection of Dependent Claims 2-9, 11-14, 17-19, 21, 25 and 27 Under 35 U.S.C. § 103

In the Office Action dated January 14, 2005, the Examiner rejected Claims 2-9, 11-14, 17-19, 21, 25 and 27 under 35 U.S.C. § 103(a) as being unpatentable over Huang, et al., (U.S. Patent No. 6,151,582) (**HU**), in view of Rumbaugh, et al. (Object oriented

modeling and Design, 1991) (**RU**), and further in view of Cowgill (U.S. Patent No. 5,835,566) (**CO**).

Applicant respectfully submits that these claims all depend from independent claims described in Section A above and contain all of the limitations of the independent claims, as amended, from which they depend. For the reasons described in Section A, **HU** and **BH** do not render those independent claims obvious and the addition of **CO** does not supply the deficiencies of that combination. Accordingly, Applicant respectfully submits that this rejection has also been traversed with respect to dependent claims 2, 9, 11-14, 17-19, 21, 25 and 27 and asks for reconsideration and allowance of those claims as well.

C. Rejection of Claim 15 Under 35 U.S.C. § 103

In the Office Action dated January 14, 2005, the Examiner rejected Claim 15 under 35 U.S.C. § 103(a) as being unpatentable over Huang, et al., (U.S. Patent No. 6,151,582) (**HU**), in view of Rumbaugh, et al. (Object oriented modeling and Design, 1991) (**RU**), and further in view of Cowgill (U.S. Patent No. 5,835,566) (**CO**) and further in view of Gunesekara (U.S. Patent No. 6,018,497) (**GU**). Like the claims discussed in section A herein, Claim 15 as amended recites in part "and a plurality of supersets of case scenarios organized in a hierarchical, non-conventional tree-like structure, having a root and one or more leaves, the tree like structure being non-conventional in that some of said case scenarios being supersets of other of said case scenarios in the tree-like structure with said supersets underlying corresponding ones of said sets in said tree like structure, such that one or more of said sets is situated between the root and the corresponding superset, ..." (Emphasis added.) For the reasons described above in Section A, a combination of **HU** and **BH** does not disclose or suggest such a flexible,

hierarchical, structure and the addition of **CO** and **GU** does not supply the deficiencies of that combination. Accordingly, Applicant respectfully submits that this rejection has also been traversed with respect to claim 15 and asks for reconsideration and allowance of claim 15 as well.

INVITATION FOR DISCUSSION

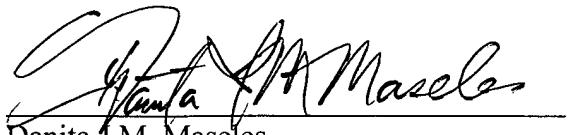
If however, the Examiner feels that it might be helpful, Applicant could arrange a telephone interview conference call, including the inventor, who lives in the United Kingdom. Applicant respectfully requests the Examiner contact Applicant if he feels such an interview would be helpful.

CONCLUSION

It is respectfully submitted that the claims of this application is in condition for allowance for the reasons stated above. Therefore, it is requested that the Examiner reconsider each and every rejection as applicable to the claims now pending in the application and pass such claims to issue.

This response is intended to be a complete response to the Office Action dated January 14, 2005.

Respectfully submitted,

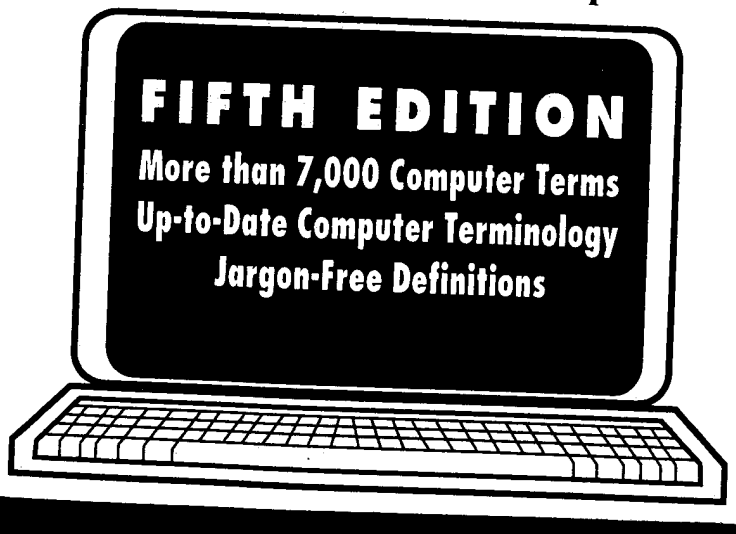


Danita J.M. Maseles,
Reg. No. 33,419
Schlumberger Information Solutions
5599 San Felipe, Suite 1700
Houston, Texas 77056
(713) 513-2515 – Office
(713) 513-2060 – Facsimile

April 13, 2005 Attorney for Applicant

Webster's New World® Dictionary of Computer Terms

Now Completely Revised & Updated



Webster's

NEW WORLD DICTIONARY® OF COMPUTER TERMS

Webster's

NEW WORLD DICTIONARY® OF COMPUTER TERMS

FIFTH EDITION

Compiled by
Donald Spencer

MACMILLAN • USA

Macmillan General Reference
A Prentice Hall Macmillan Company
15 Columbus Circle
New York, NY 10023

A Webster's New World™ Book

Dictionary Editorial Offices:
New World Dictionaries
850 Euclid Avenue
Cleveland, Ohio 44114

Library of Congress Cataloging-in-Publication Data

Spencer, Donald.

ISBN: 0-671-89993-7

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Fifth Edition

Introduction

It is unlikely that any field has contributed more new terms (or new meanings of old ones) to the language in the last few years than has computer technology. This is largely because computer technology is itself a new and changing field. As it evolves, fresh terminology must be developed to communicate, describe, and define the heretofore unknown concepts, components, and techniques. This book provides a glossary of over 7,000 words, phrases, acronyms, and abbreviations used in connection with computers.

When a business installs its first computer, not only are the owners and managers apprehensive; the majority of the office and administrative personnel are also involved in the change that has taken place. Managers must know something about the current computer jargon in order to assist the staff with knowledgeable explanations of what is happening. Filling the need for this information is the purpose of this book.

This new edition offers up-to-date coverage of terms used in reference to hardware, software, programming, logic, and computer graphics, as well as those used in ancillary fields such as data communications and artificial intelligence. Areas of frequent application are also covered, particularly mathematics and business administration.

The terms in this book have been selected as those most likely to confront the beginning computer-user, and they are defined in a way that any lay person can understand. Wherever possible, technical terms have been avoided so that the definitions might be easily read and understood. Where

a CAD system for visual verification and editing prior to final output generation.

checkpoint A specified point at which a program can be interrupted, either manually or by a control routine. Used primarily as an aid in debugging programs.

check problem A testing problem designed to determine whether a COMPUTER or a COMPUTER PROGRAM is operating correctly.

check sum The summation of digits or bits used primarily for checking purposes and summed according to an arbitrary set of rules. Used to verify the integrity of data.

chiaroscuro Shading.

chiclet keyboard A type of microcomputer keyboard. The name was derived from the chewing gum because the keys are small and square, resembling the gum pieces. Chiclet keys are very small and typically spread out, so that touch typing is more difficult than on a conventional keyboard. An undesirable keyboard used in older low-cost microcomputers; however, a useful keyboard in systems for children and handicapped people.

chief information officer The manager of an MIS department.

chief programmer team A software development team structure in which one programmer is assigned overall responsibility for the project.

child A data record that can be created only based upon the contents of one or more other records (parents) already in existence.

children In hierarchical databases, those data elements that are linked in a subordinate fashion to a parent element.

chip A small component that contains a large amount of electronic circuitry; a thin silicon wafer on which electronic components are deposited in the form of integrated circuits. Chips are the building blocks of a computer and perform various functions, such as doing arithmetic, serving as the computer's memory, or controlling other chips.

Leaders are sometimes used in tables of contents to lead the reader's eye from the entry to the page number. **(2)** The blank section of tape at the beginning of a reel of magnetic tape.

leading The vertical spacing between lines of type, measured from baseline to baseline. Font styles that have long ascenders and descenders need more leading than fonts that don't. In publishing, the font size and leading is described as a fraction. For example, 10/12 (which is read "10 on 12") indicates 10-point type with two points of leading.

leading edge **(1)** In OPTICAL SCANNING, the edge of the document or page that enters the read position first. **(2)** A buzz word implying technological leadership, as in "the leading edge of technology."

leaf The terminal node of a tree diagram. See ROOT.

learning A procedure in ARTIFICIAL INTELLIGENCE by which an artificial intelligence program improves its performance by gaining knowledge.

learning machine A machine that can monitor its own behavior and employ feedback to modify that behavior.

lease A method of acquiring the use of a computer system. A lease contract requires no financing and is less expensive than renting the system.

leased line A communications line dedicated to one customer. Also called private line.

leasing companies Companies that specialize in leasing computer equipment, which they purchase from a computer manufacturer.

least significant digit (LSD) That digit of a number that has the least weight or significance. In the number 58,371, for example, the least significant digit is 1.

LED Acronym for **l**ight-**e**mitting **d**iode, a commonly used alphanumeric display unit that glows when supplied with a specified voltage.

LED printer Acronym for **l**ight-**e**mitting **d**iode printer, an electrophotographic printer that uses an electrostatically charged drum to transfer toner to a piece of paper. Similar to LASER PRINTER and LCS PRINTER.

hierarchical model A database model in which each object is of a particular hierarchy in a tree structure.

hierarchical network A computer network in which processing and control functions are performed at several levels by computers specially designed for the functions performed.

hierarchical structure In DATABASE MANAGEMENT SYSTEMS, the simplest form of file organization, in which records of various levels are related by owning or belonging to each other.

hierarchy **(1)** The order in which arithmetic operations within a formula or statement will be executed. **(2)** Arrangement into any graded series.

hierarchy chart A chart that divides the software package into small pieces until a program can be written to implement each piece.

hierarchy plus input-process-output (HIPO) A design and program documentation method that represents functional structure and data flow in a series of three types of diagrams: visual tables of contents that name the program modules and specify their hierarchical relationships; overview diagrams that describe the input, processing, and output for members of the hierarchy; and detailed diagrams that extend the overview diagrams to include more specific input, processing, and output detail with narrative.

high-density disk A floppy disk that holds more information than a double-density disk. A 5.25-inch high-density disk holds 1.2 megabytes, and a 3.5-inch high-density disk holds 1.44 megabytes.

high-end High-priced.

high-level language Any programming language that allows users to write instructions in a familiar notation rather than in a machine code. Each statement in a high-level language corresponds to several machine-code instructions. Contrast with LOW-LEVEL LANGUAGE.

high-level programming language See HIGH-LEVEL LANGUAGE.

highlight **(1)** In graphics, refers to a user's ability to enhance a graph or

cuit simulation, it can be used in the engineering prototype or reproduction model to find and correct program errors or in the production model to add new features.

root The top element or node in a tree diagram, from which branches extend eventually to LEAF nodes.

root directory Under DOS, the first, top-level directory.

roping An aliasing effect in a graphics image in which a line or thin polygon appears to vary in width, color, or brightness according to a repeating pattern that suggests the braiding of a rope.

rotating memory A magnetic information storage device in the form of a round platter that is spun like a phonograph record. See MAGNETIC DISK.

rotation In computer graphics, the turning of a computer-modeled object relative to an origin point on a coordinate system. In three-dimensional graphics, an object can be rotated in space, usually around the axis, to provide different views. See TRANSFORMATION.

rotational delay The time it takes for a record contained on one of the sectors of a disk to rotate under the read/write head.

rote learning A type of learning in which all knowledge is explicitly provided by an external source, as by programmers in conventional computer programming.

rough draft In DESKTOP PUBLISHING, the preliminary page layouts done by the designer using pencil sketches to represent page design ideas; a roughly drawn sketch of a finished document.

round See ROUND OFF.

rounding The process of dropping the least significant digit or digits of a numeral, and adjusting the remaining numeral to be as close as possible to the original number.

rounding error An error due to ROUNDING.

round off To truncate the rightmost digit of a number, and to increase by

visible until the colors are switched. **(3)** A line of a diagram that is invisible.

hidden line removal The process of deleting line segments from a drawing when they would be obscured if the object were displayed as a solid three-dimensional figure. Many types of computer graphics software and hardware can remove such hidden lines automatically.

hidden object Distinct graphic entities that would be obscured from view by other entities if they were displayed as solids.

hidden surface An entire surface or plane that would be obscured from view if the graphics figure were displayed as a three-dimensional solid.

hide To remove a window and all its associated windows from a desktop.

hierarchical database management system (HDBMS) A collection of related programs for loading, accessing, and controlling a database in which the data are organized like an inverted tree with a series of nodes connected by branches.

hierarchical data model A data model in which each element has only one parent or owner.

hierarchical data structures In a Programmer's Hierarchical Interactive Graphics System, objects are defined in hierarchical relationship to one another. The hierarchical data organization allows structures to inherit the attributes of other structures, enabling the programmer to manipulate objects efficiently.

hierarchical directories A term used to refer to the organizational method of arranging files either in a DOS tree structure or in the file-and-folder method of Apple Macintosh computers.

hierarchical file system (HFS) A pyramid-like file system in which each object is linked to those beneath it.

hierarchical local network A star-shaped local network in which a large central processing unit is at the top of the hierarchy and communications terminals or small CPUs are at the bottom.

Prior Art

Object-Oriented Modeling and Design

James Rumbaugh
Michael Blaha
William Premerlani
Frederick Eddy
William Lorensen

General Electric Research and Development Center
Schenectady, New York

PRENTICE HALL
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Object-oriented modeling and design / James Rumbaugh ... [et al.].
p. cm.
Includes bibliographical references and index.
ISBN 0-13-629841-9
1. Object-oriented programming (computer science) 2. System design. I. Rumbaugh, James.
QA76.64.O26 1991
005.1--dc20 90-7600
CIP

Editorial/production supervision: *Kathleen Schiaparelli*
Cover design: *Butler/Udell Design*
Manufacturing buyer: *Linda Behrens/Patrice Fraccio*

A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

Printed in the United States of America

20 19 18 17 16 15 14 13 12

ISBN 0-13-629841-9

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Conte

Aggregation is drawn like association, except a small diamond indicates the assembly end of the relationship. Figure 3.21 shows a portion of an object model for a word processing program. A document consists of many paragraphs, each of which consists of many sentences.

Figure 3.21 Aggregation

The existence of a component object may depend on the existence of the aggregate object of which it is part. For example, a binding is a part of a book. A binding cannot exist apart from a book. In other cases, component objects have an independent existence, such as mechanical parts from a bin.

Figure 3.22 demonstrates that aggregation may have an arbitrary number of levels. A microcomputer is composed of one or more monitors, a system box, an optional mouse, and a keyboard. A system box, in turn, has a chassis, a CPU, many RAM chips, and an optional fan. When we have a collection of components that all belong to the same assembly, we can combine the lines into a single aggregation tree in the diagram. The aggregation tree is just a shorthand notation that is simpler than drawing many lines connecting components to an assembly. An object model should make it easy to visually identify levels in a part hierarchy.

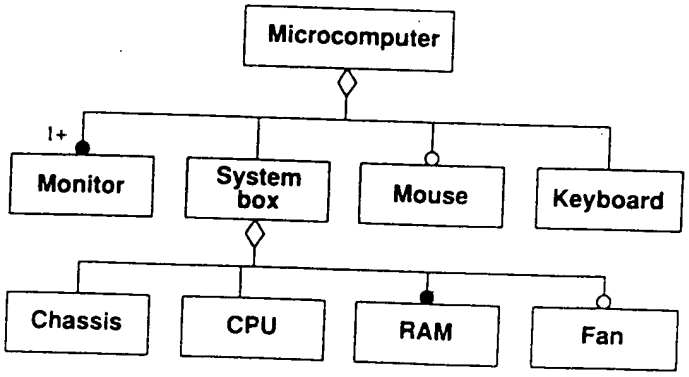


Figure 3.22 Multilevel aggregation

3.4 GENERALIZATION AND INHERITANCE

3.4.1 General Concepts

Generalization and inheritance are powerful abstractions for sharing similarities among classes while preserving their differences. For example, we would like to be able to model the following situation: Each piece of equipment has a manufacturer, weight, and cost.

Pumps also have suction pressure and flow rate. Tanks also have volume and pressure. We would like to define equipment features just once and then add details for pump, tank, and other equipment types.

Generalization is the relationship between a class and one or more refined versions of it. The class being refined is called the *superclass* and each refined version is called a *subclass*. For example, *Equipment* is the superclass of *Pump* and *Tank*. Attributes and operations common to a group of subclasses are attached to the superclass and shared by each subclass. Each subclass is said to *inherit* the features of its superclass. For example, *Pump* inherits attributes *manufacturer*, *weight*, and *cost* from *Equipment*. Generalization is sometimes called the "is-a" relationship because each instance of a subclass is an instance of the superclass as well.

Generalization and inheritance are transitive across an arbitrary number of levels. The terms *ancestor* and *descendent* refer to generalization of classes across multiple levels. An instance of a subclass is simultaneously an instance of all its ancestor classes. The state of an instance includes a value for every attribute of every ancestor class. Any operation on any ancestor class can be applied to an instance. Each subclass not only inherits all the features of its ancestors but adds its own specific attributes and operations as well. For example, *Pump* adds attribute *flow rate*, which is not shared by other kinds of equipment.

The notation for generalization is a triangle connecting a superclass to its subclasses. The superclass is connected by a line to the apex of the triangle. The subclasses are connected by lines to a horizontal bar attached to the base of the triangle. For convenience, the triangle can be inverted, and subclasses can be connected to both the top and bottom of the bar, but if possible the superclass should be drawn on top and the subclasses on the bottom.

Figure 3.23 shows an equipment generalization. Each piece of equipment is a pump, heat exchanger, tank, or another type of equipment. There are several kinds of pumps: centrifugal, diaphragm, and plunger. There are several kinds of tanks: floating roof, pressurized, and spherical. *Pump type* and *tank type* both refine second level generalization classes down to a third level; the fact that the tank generalization symbol is drawn below the pump generalization symbol is not significant. Several object instances are displayed at the bottom of the figure. Each object inherits features from one class at each level of the generalization. Thus *P101* embodies the features of equipment, pump, and diaphragm pump. *E302* assumes the properties of equipment and heat exchanger.

The dangling subclass ellipsis (triple dot) in Figure 3.23 indicates that there are additional subclasses that are not shown on the diagram, perhaps because there is no room on the sheet and they are shown elsewhere, or maybe because enumeration of subclasses is still incomplete.

The words written next to the triangles in the diagram, such as *equipment type*, *pump type*, and *tank type*, are discriminators. A *discriminator* is an attribute of enumeration type that indicates which property of an object is being abstracted by a particular generalization relationship. Only one property should be discriminated at once. For example, class *Vehicle* can be discriminated on propulsion (wind, gas, coal, animal, gravity) and also on operating environment (land, air, water, outer space). The discriminator is simply a name for the basis of generalization. Discriminator values are inherently in one-to-one correspondence with the subclasses of a generalization. For example, the operating environment discriminator for

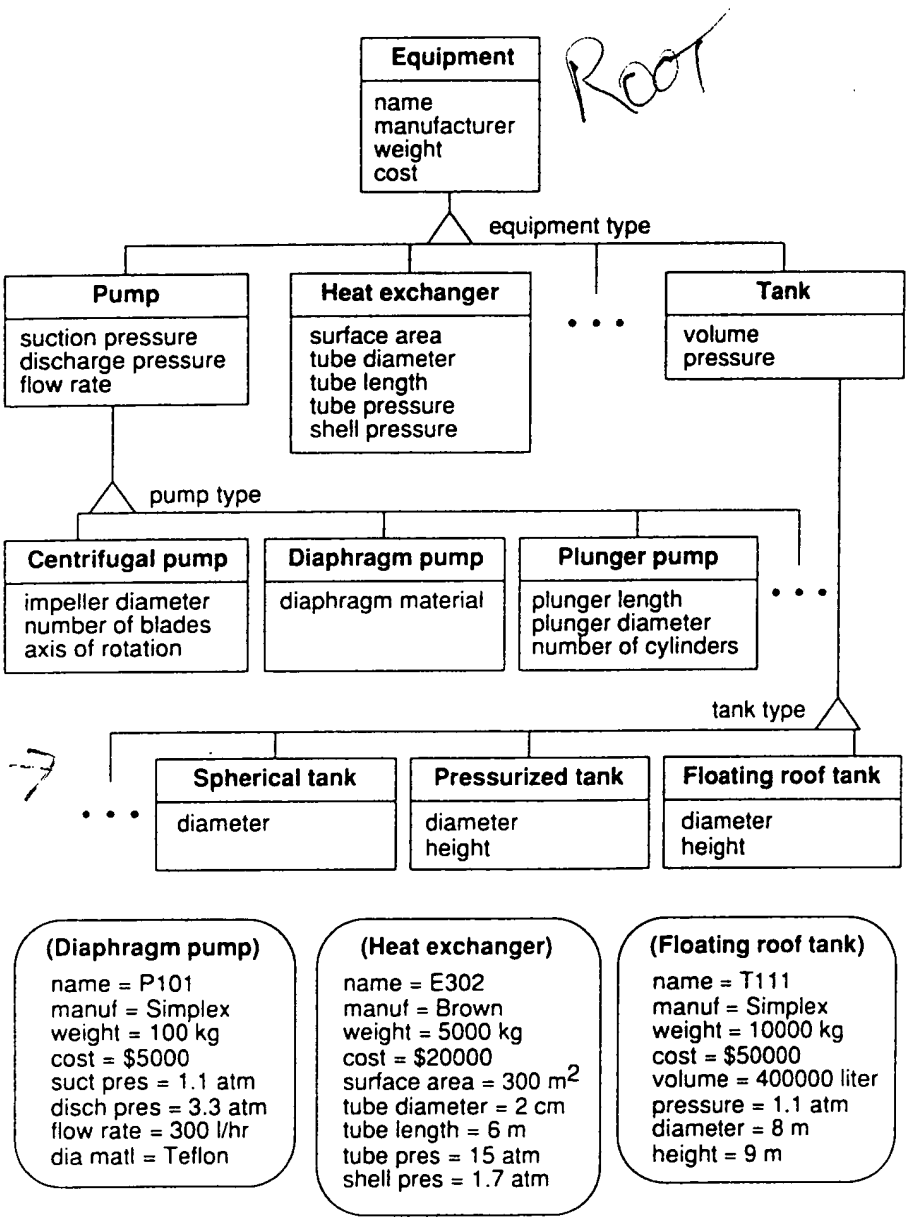


Figure 3.23 A multilevel inheritance hierarchy with instances

Boat is *water*. The discriminator is an optional part of a generalization relationship; if a discriminator is included, it should be drawn next to the generalization triangle.

Figure 3.24 shows classes of graphic geometric figures. This example has more of a programming flavor and emphasizes inheritance of operations. *Move*, *select*, *rotate*, and *display*

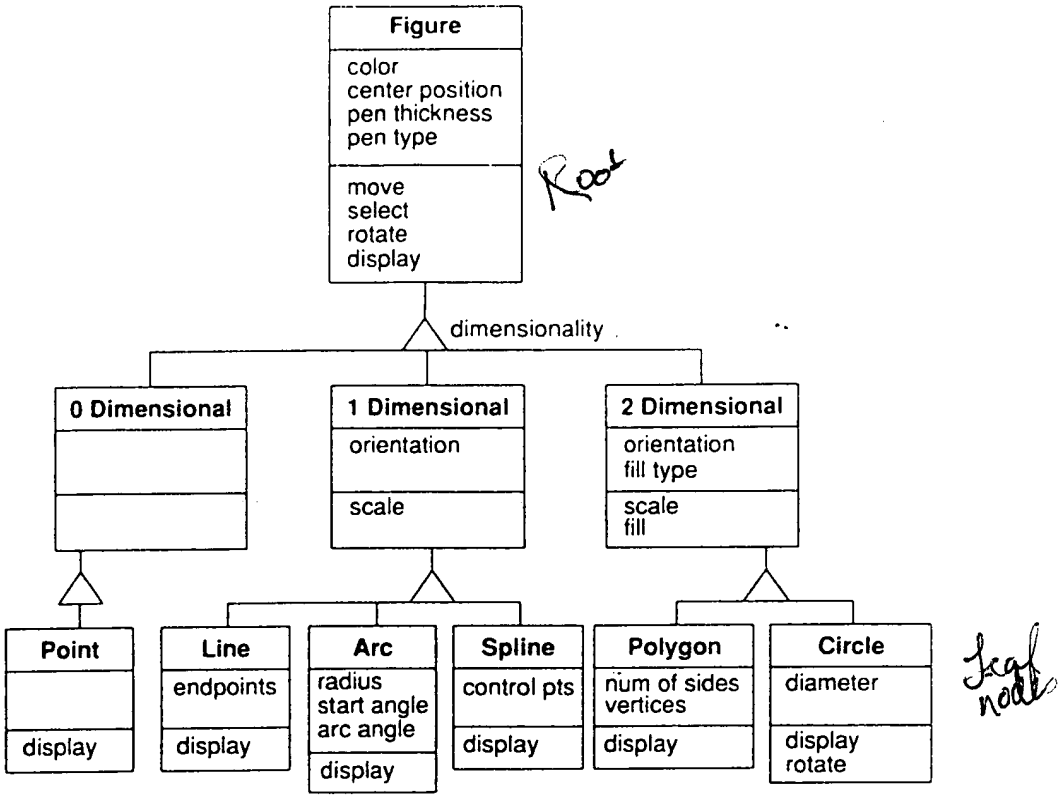


Figure 3.24 Inheritance for graphic figures

are operations inherited by all subclasses. *Scale* applies to one- and two-dimensional figures. *Fill* applies only to two-dimensional figures.

Do not nest subclasses too deeply. Deeply nested subclasses can be difficult to understand, much like deeply nested blocks of code in a procedural language. Often with some careful thought and a little restructuring, you can reduce the depth of an overextended inheritance hierarchy. In practice, whether or not a subclass is "too deeply nested" depends upon judgment and the particular details of a problem. The following guidelines may help: An inheritance hierarchy that is two or three levels deep is certainly acceptable; ten levels deep is probably excessive; five or six levels may or may not be proper.

3.4.2 Use of Generalization

Generalization is a useful construct for both conceptual modeling and implementation. Generalization facilitates modeling by structuring classes and succinctly capturing what is similar and what is different about classes. Inheritance of operations is helpful during implementation as a vehicle for reusing code.

Object-oriented languages provide strong support for the notion of inheritance. (The concept of inheritance was actually invented far earlier, but object-oriented languages made it popular.) In contrast, current database systems provide little or no support for inheritance. Object-oriented database programming languages (Section 15.8.5) and extended relational database systems (Section 17.4) show promise of correcting this situation.

Inheritance has become synonymous with code reuse within the object-oriented programming community. After modeling a system, the developer looks at the resulting classes and tries to group similar classes together and reuse common code. Often code is available from past work (such as a class library) which the developer can reuse and modify, where necessary, to get the precise desired behavior. The most important use of inheritance, however, is the conceptual simplification that comes from reducing the number of independent features in a system.

The terms inheritance, generalization, and specialization all refer to aspects of the same idea and are often used interchangeably. We use *generalization* to refer to the relationship among classes, while *inheritance* refers to the mechanism of sharing attributes and operations using the generalization relationship. Generalization and specialization are two different viewpoints of the same relationship, viewed from the superclass or from the subclasses. The word *generalization* derives from the fact that the superclass generalizes the subclasses. *Specialization* refers to the fact that the subclasses refine or specialize the superclass. In practice, there is little danger of confusion.

3.4.3 Overriding Features

A subclass may *override* a superclass feature by defining a feature with the same name. The overriding feature (the subclass feature) refines and replaces the overridden feature (the superclass feature). There are several reasons why you may wish to override a feature: to specify behavior that depends on the subclass, to tighten the specification of a feature, or for better performance. For example, in Figure 3.24, *display* must be implemented separately for each kind of figure, although it is defined for any kind of figure. Operation *rotate* is overridden for performance in class *Circle* to be a null operation. Chapter 4 discusses overriding features in more detail.

You may override default values of attributes and methods of operations. You should never override the *signature*, or form, of a feature. An override should preserve attribute type, number, and type of arguments to an operation and operation return type. Tightening the type of an attribute or operation argument to be a subclass of the original type is a form of *restriction* (Section 4.3) and must be done with care. It is common to boost performance by overriding a general method with a special method that takes advantage of specific information but does not alter the operation semantics (such as *rotate-circle* in Figure 3.24).

A feature should never be overridden so that it is inconsistent with the signature or semantics of the original inherited feature. A subclass *is* a special case of its superclass and should be compatible with it in every respect. A common, but unfortunate, practice in object-oriented programming is to "borrow" a class that is similar to a desired class and then modify it by changing and ignoring some of its features, even though the new class is not really a